EXHIBIT 1

XMC Web Models

Community and Scheduling

Author: Dave Brown
Create Date: February 22, 2000 Save Date: XXX 0, 0000 Print Date: February 25, 2000
Project: $/prjcmpnt/xmc/v.100/persdev/doc/_devpers/web_models/webmodels
Document: Document2
Description:
Revisions:

ROY-G-BIV®
Software for a spectrum of ideas™

Table of Contents

1 • Overview

This document describes web models that are used to serve content, such as motion control instructions, audio, video, computer instructions and other media, in ways that are useful in that they allow users to collaborate with others when creating and using content as well as schedule times when content is to be played on content players.

There are three main chapters in this document that describe networked content models that allow a group of content users to collaborate as a community as well as networked content models that allow users to schedule when content is played. The chapters in this document are as follows:

- **Chapter 1 - Overview**; this chapter.
- **Chapter 2 - Community Model**; describes the network community model that allows several content users and/or creators collaborate on the creation or use of content.
- **Chapter 3 - Scheduling Model**; describes the network scheduling model that allows user to schedule when content is to be played.

2 • Community Model

The community model is designed to be a meeting place where more than one user can collaborate, share and discuss content. Users upload content that they create for use by (or sale to) others. Other users may merely download (or purchase) new content that others create for use on their local players where a player may be a toy, or other electronic device capable of running the content.

The following diagram shows the community model.

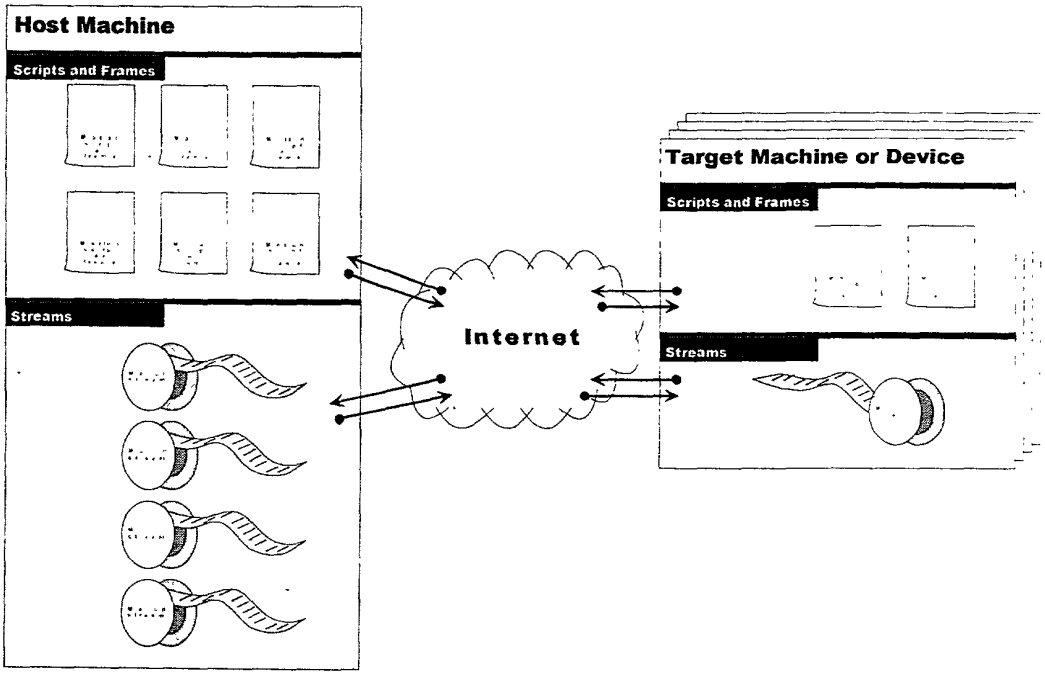


Figure 1 Community Model.

When using this content model users may either work as an individual where content is uploaded and downloaded across a network (ie the Internet, a local network, etc.) by a single person, or the model may be used by a group where each action performed when running the content is synchronized among all members of the group. The following sections describe each of these ideas in more detail.

Individual Sessions

Individual sessions involve a single user who downloads content for playing and/or uploads content that they create. To play content, the user browses the content list via a directory list, web browser, or other means of displaying the titles of each content script or content stream that is to be played.

The user may also create content of their own that they wish to share with others. Once created, the user uploads their content to the content site on the network.

Group Sessions

Group sessions use similar methods of uploading and downloading as described in the individual sessions with a new addition – users in the group are able to collaborate with one another in real-time. For example a set of users may optionally choose to run the same content script or stream. Using content synchronization, described below, the content running on each users machine or device is synchronized thus giving each end user the same experience.

NOTE: *when combined with scheduling, a group of devices may be synchronized with one other without requiring the intervention of the user. The chapter on the Scheduling Model below describes this in more detail.*

Content Synchronization

Sometimes is useful to have each machine or device play the same content script or stream and remain in sync with one another. For example, two users may own a toy doll that when directed to play a certain content script, both dolls sing and dance at the same time. In order to give each user a similar experience, both user devices are synchronized with one another either by the host machine or by communicating with one another. When synchronized each device runs the same content at the same time. If both dolls were placed side by side they would dance in a synchronized fashion when running the same content. It is not as important that each device run the same content, but when they do run the actions run are run in a manner that is in sync with the other device.

Host-to-Device synchronization is a synchronization model driven by the host (which could easily be a target device playing the host roll), where the host broadcasts content to other target devices. While broadcasting, the content data is injected with synchronization packets that each device uses to adjust the rate in which they play the content data received. Usually the host-to-device model is used when running a stream of content data on many target devices.

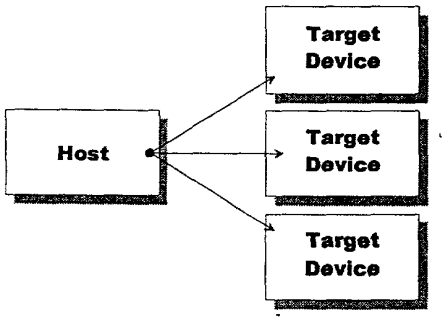


Figure 2 Host-to-Device Model.

When using the device-to-device synchronization model, a certain device requests that another device start a synchronization session. During the synchronization session, both devices periodically send synchronization packets to one another thus allowing each to change their individual play rates accordingly. Usually device-to-device synchronization is used when each device plays a content script that has been downloaded from the host.

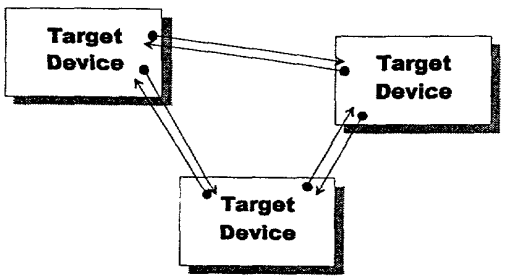


Figure 3 Device-to-Device Model.

Host-to-Device Synchronization

When using host-to-device synchronization, the host machine generates synchronization packets and injects them into the stream that is being played by each of the target machines.

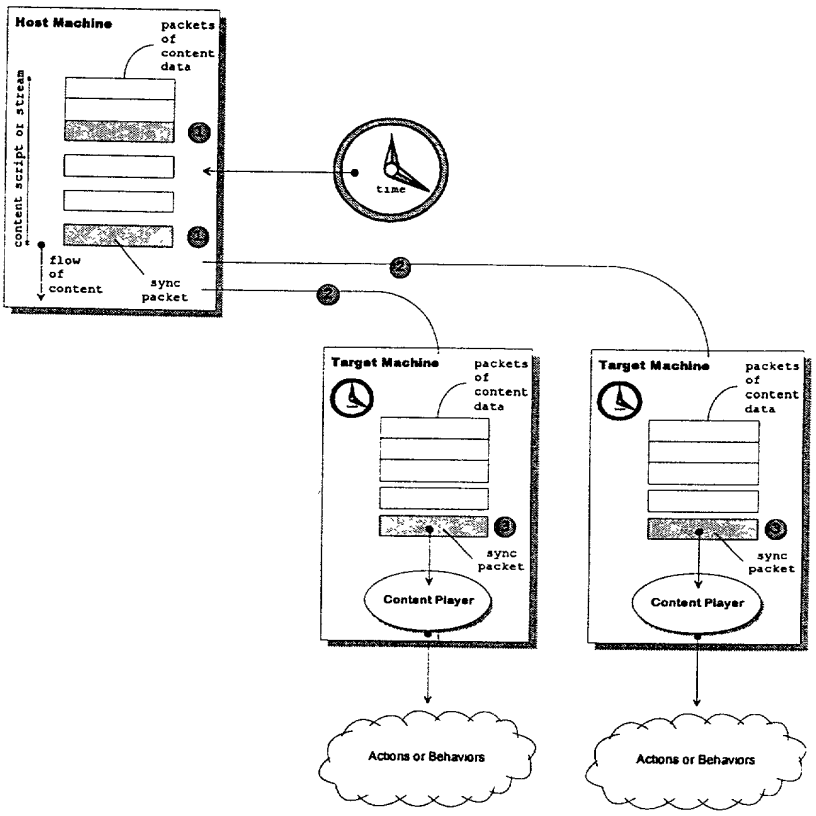


Figure 4 Host to Device Synchronization.

The following steps occur during host-to-device synchronization.

Step	Description
1	Using either a time signature, a packet sequence number, or some other reference data, the host periodically builds each synchronization packet. Each synchronization packet is then injected at pre-defined intervals into the content data.
2	The content data is sent to one or more target devices in a broadcast fashion where all data is sent out to all at the same time or as close to the same time as possible using a round robin approach where each packet is sent to all targets before the next packet is sent out.
3	Upon receiving each packet, the target devices buffer each packet until the synchronization packet is received. Upon receiving the synchronization packet, the remaining packets are processed by the content player thus causing movement or other actions to occur.

Device-to-Device Synchronization

When using device-to-device synchronization one device sends a synchronization packet to another requesting that it run a script or stream in sync with the requesting device.

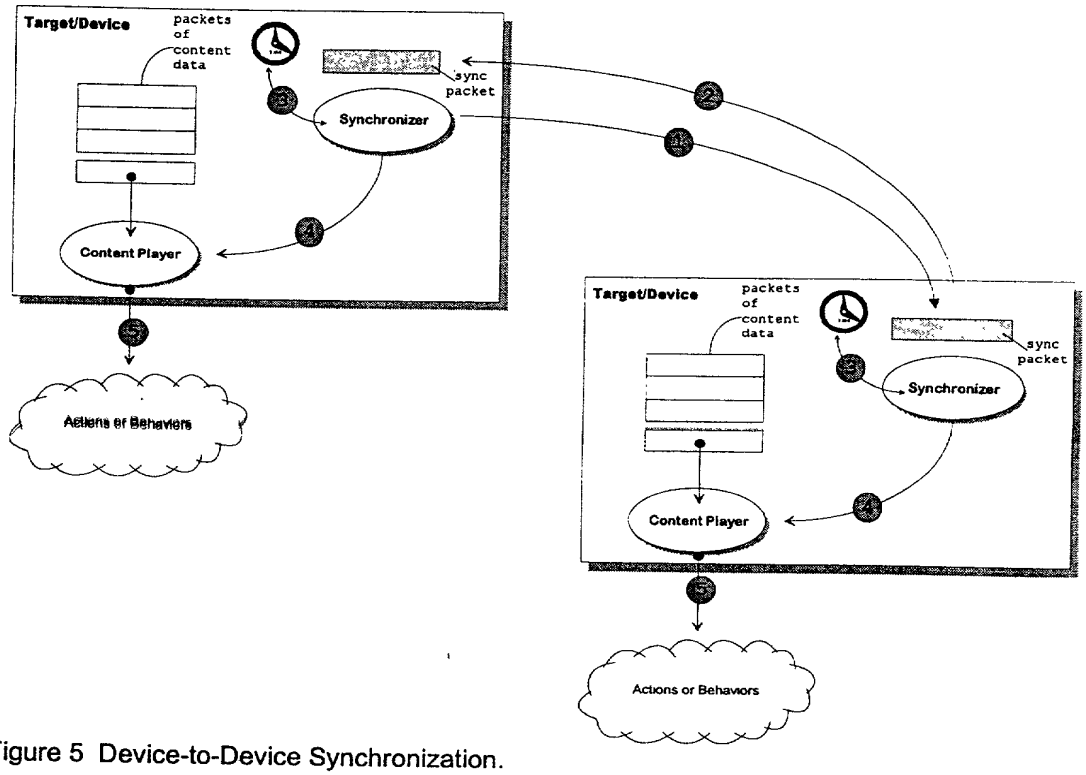


Figure 5 Device-to-Device Synchronization.

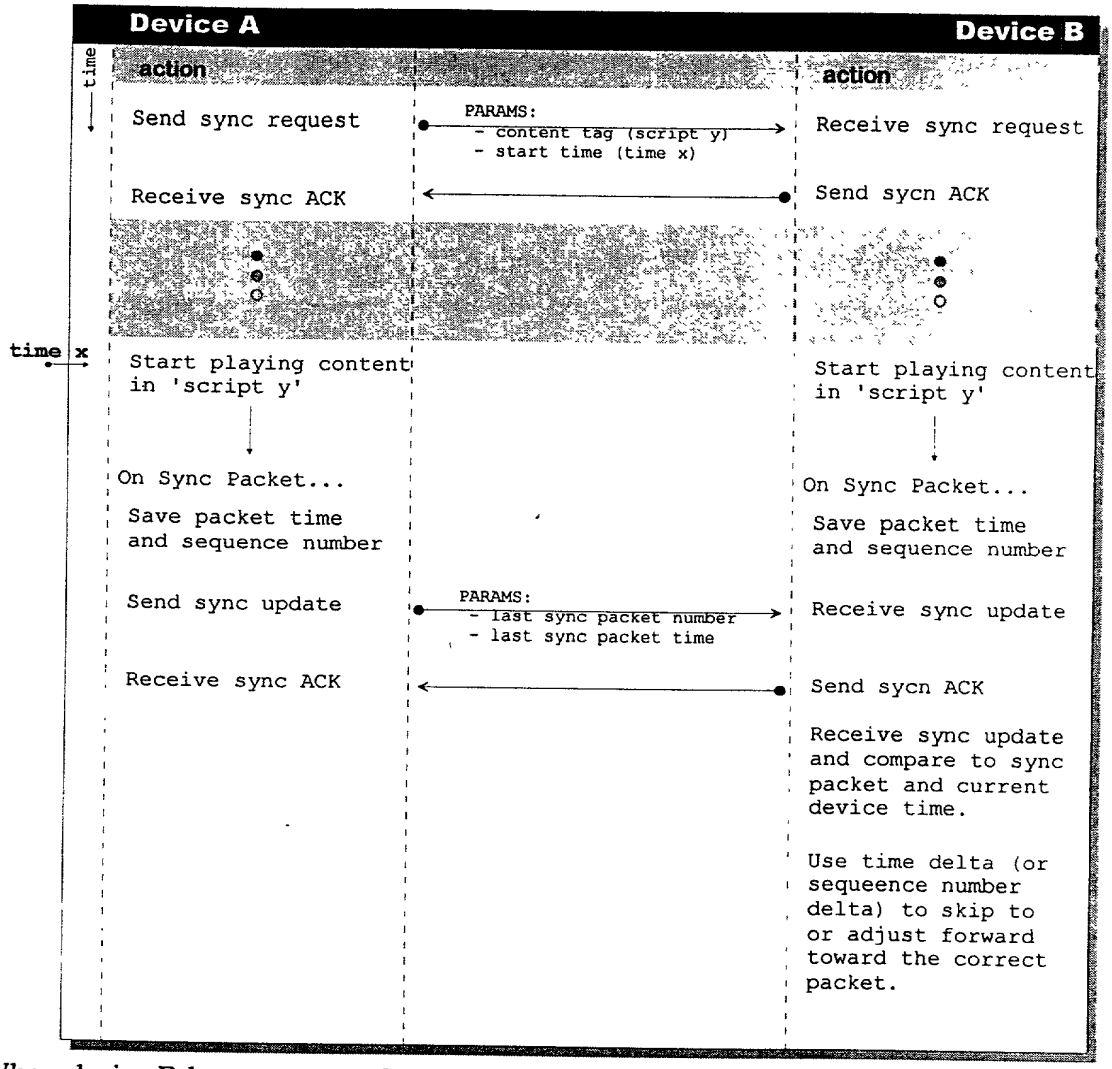
When performing device-to-device synchronization, the following steps occur.

Step	Description
1	To start the sync process, a device requests another device to start a synchronization session. In the request, the original device sends the script or stream name (that is to be run) and the start time.
2	Upon receiving the sync request, the receiving device responds with an acknowledgement message (ACK).
3	Each device waits for the start time, specified in the sync request, to strike.
4	When the time hits, both devices direct their content players to start running the content.
5	Running the content causes motions and other actions to occur.

NOTE: *Device-to-device synchronization assumes that all device clocks have been synchronized at least once with an outside reference point. For example, all devices may be requested (by a central host) to update their internal clocks with a GMT web server, or with the central host machine's clock.*

Synchronization Handshaking

The following shows an example synchronization handshaking session where Device A first requests that Device B synchronize with it at a certain time **X** with script **Y**.



When device B becomes out of sync with device A, it must either adjust its play rate of content packets or if the data contains motion, adjust the velocity of the moves (if any) caused by running the content data.

Play Rate Synchronization

The general algorithm used to decide how to adjust the play rate is as follows:

```
TimeA  = time of sync packet from deviceA
TimeB  = time of sync packet from deviceB
Tdelta = TimeA - TimeB

If (Tdelta > 0)
    Increase the Play Rate

Else If (Tdelta < 0)
    Decrease the Play Rate

Else
    Don't change the Play Rate
```

There are many ways to adjust the play rate and usually the method chosen will depend on the type of content that is used.

If the content contains dimensional point data (such as **<x,y,z>** for thee dimensional points), new points may be inserted within the point set thus causing each move to take slightly more time thus slowing down the play rate. In the same example speeding up the play rate is accomplished by skipping point data.

In another example, if the content contains motion instructions that involve causing moves at a certain velocity; the actual velocities may be altered causing the move directed by the instruction to complete in a shorter or longer amount of time. For example, to increase the play rate with this method, the velocity of a move would be increased slightly, which would in turn cause the motion instruction to complete in a shorter amount of time. Usually move instructions are accompanied with a 'wait for move' instruction which causes the instruction processor to wait until the move is complete before running the next instruction.

3 • Scheduling Models

Scheduling is used both direct the host to start broadcasting content and/or the device to start running content that is already downloaded or being broadcast. Host scheduling involves scheduling times where the host machine is to carry out certain actions such as initiating a broadcast secession, etc. Target scheduling involves scheduling each target device to begin running content at certain pre-determined points in time.

Host Scheduling and Broadcasting

With host scheduling, the host machine is configured to carry out certain operations at pre-determined points in time. For example, much like Television programming, the host machine may be configured to schedule broadcasting certain content streams at certain points in time.

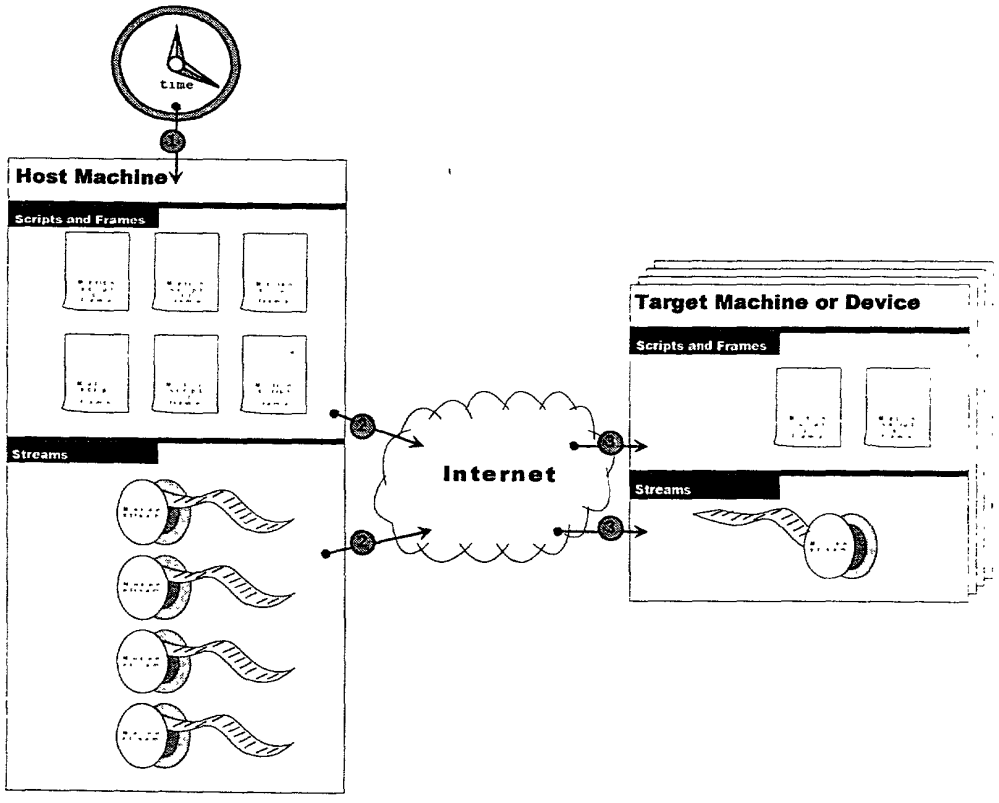


Figure 6 Host Scheduling Model

The following steps occur during host scheduling.

Step	Description
1	At certain pre-defined points in time, the host start performing the pre-determined actions such as broadcasting stream data.
2	At the pre-determined times, the host starts broadcasting content to the network (ie internet, or internal network).
3	Target devices that are configured to 'tune-into' certain broadcast channels run the content as it is received.

As an example of host scheduling, a host machine may be configured with several content streams that contain interleaved motion/audio data. At each pre-determined 'scheduled' time, the host starts broadcasting the interleaved motion/audio data to any devices that may be listening. As an example, the listening device may be a dancing mannequin that plays music as it dances thus giving the appearance that the mannequin is dancing to the music.

Target Scheduling

Target scheduling involves the target device being programmed to request and run content from the host (or run data sets already downloaded) at certain scheduled times.

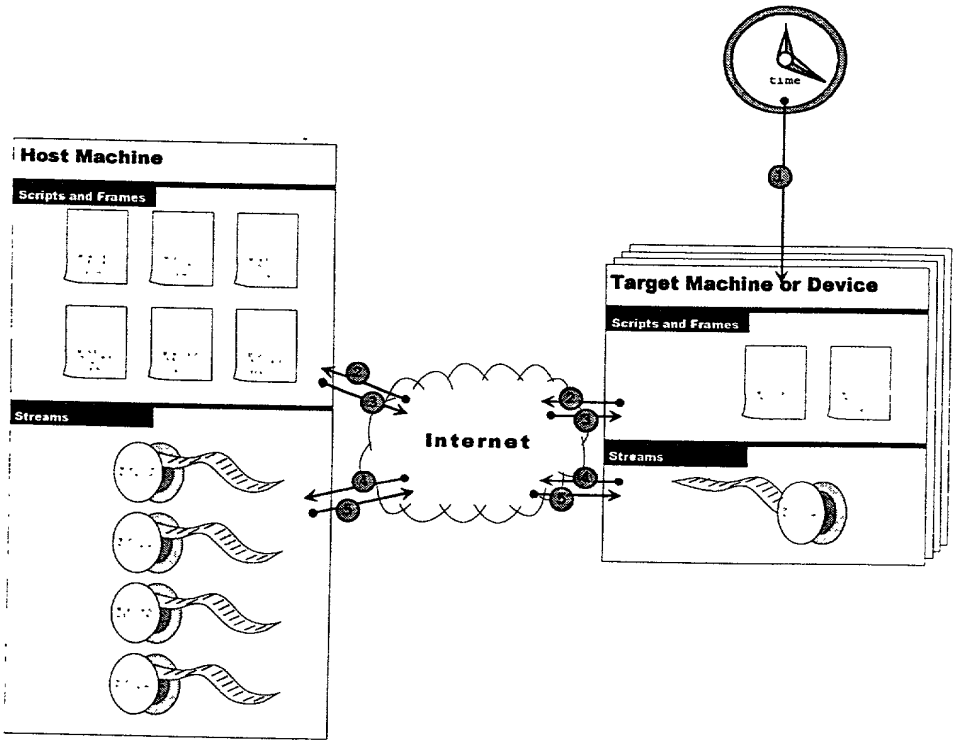


Figure 7 Target Scheduling Model.

The following steps occur during target based scheduling.

Step	Description
1	The target device is programmed to wait for a certain point in time to hit.
2	When the scheduled time hits, the target device's scheduler wakes up and queries the host across the network for a content script.
3	Upon receiving the content script, the target device begins running the content.
4	When the scheduled time hits, the target device's scheduler may also query the host across the network for a content stream.
5	Upon receiving the content stream, the target device starts running each instruction.

As an example of target based scheduling, the owner of a motion-based toy would go to a web site and select a certain motion script or stream to run on a certain data (i.e. a parent may select the Macarena dance and run it on their child's birthday as a surprise). Once scheduled, on the child's birthday, the toy would automatically connect to the host web site, download the data and start running the dance.

EXHIBIT 2

XMC Extensions

Scripts and Framing of Motion Sequences

Author: Dave Brown
Create Date: October 26, 1999 Save Date: XXX 0, 0000 Print Date: October 27, 1999
Project: $/prjcmpnt/xmc/v.100/int/persdev/doc/des/script_frames
Document: Document2
Description:
Revisions:

ROY-G-BIV®
Software for a spectrum of ideas™

Table of Contents

1 • Overview

Many applications that use motion control often use several sequences of basic motion operations to perform the operations carried out by the machine. The creation of each motion sequence is usually created off-line and then downloaded to the machine on which it is run. This model becomes fairly limited, and almost unusable, when the connection between the host machine (used to create and store the motion sequence) and the target device (the consumer of the motion data) becomes intermittent or unreliable.

This document describes the process of breaking up each sequence of basic motion operations, or motion scripts, into small frames of motion operations that must be sent in their entirety before actually being run on the target device. Breaking a script of motion operations into small frames is very important for it is easier to send small data packets across an unreliable or intermittent data line than it is to send large data packets.

2 • System Design

The meta scripting system is made up of scripts, frames and meta commands, where each frame is a set of meta commands and each script is a set of motion frames.

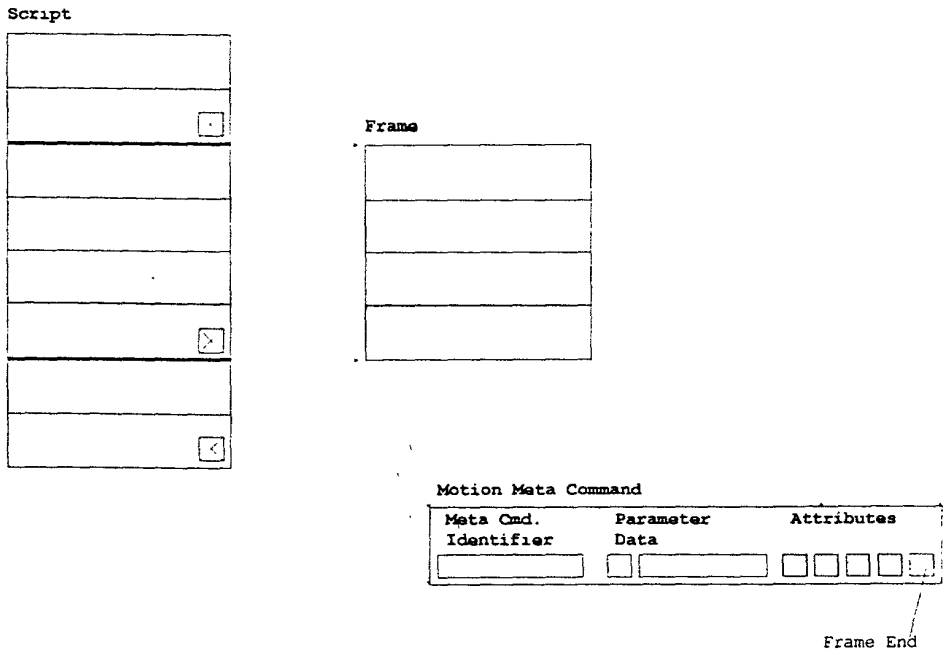
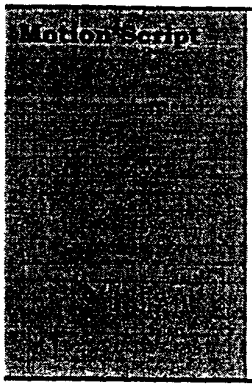


Figure 1 Meta Scripting System

The following pieces make up the motion script and framing system.

Item	Description
[redacted]	A meta command is an atomic script element that describes a simple motion control operation that is to be executed by the target device. A motion frame is a set of meta commands that end with a meta command that has its 'Frame End' attribute set to TRUE. All other previous meta commands in the frame must have this attribute set to FALSE. The script designer is recommended to build each frame making up a script in such a manner that it will run successfully on the device in its entirety and place the device in a safe state upon its completion. For example the designer should end all frames that contain move operations with a Wait operation that waits for all motion to complete) before continuing processing other motion operations located within another frame.



A motion script is a set of motion frames that define a certain operation carried out by the target device. The script will execute one frame at a time on the device.

Constructing the script with frames of meta commands is recommended for doing so allows the designer to somewhat detach the download of the script to the device from the execution of those scripts on the target device. For example, while the host is downloading a large script, the target device may actually start running the first frames in the script even before receiving the entire script. This is an option that the device may elect to take, but is not required.

Meta Commands

Each meta command contains all information necessary to describe a basic motion operation carried out by the script engine running on the target device. The script engine may opt to use one of many different forms of implementations. For example, the hardware independent XMC Motion Control system may be used as an implementation, or the script engine may just as well use a hardware dependent implementation for each operation. The meta command data allows the designer to separate the data describing the motion control operation from the actual implementation.

Definition

```
typedef struct XMCMetaCommand
{
      DWORD        dwMetaCmdID;
      LPVARIANT    pParamData;
      DWORD        dwParamDataCount;
      DWORD        dwFlags;
};
```

dwMetaCmdID – The Meta command identifier, which is a unique identifier that corresponds to a certain basic motion operation implementation that is to be run when this meta command is encountered by the script engine running on the target device.

pParamData – array of VARIANT [1]structures that each describe a single parameter used when running the meta command.

dwParamDataCount – number of elements within the *pParamData* array of elements.

dwFlags – set of attribute flags that describe the meta command and how it is to be processed. The following flags are supported.

Flag	Description
XMC_MF_FRAMEEND	End meta command within the current

[1] For more information on the VARIANT structure see the Microsoft® Win32® Reference Guide or the Microsoft® MSDN® on-line help.

frame of commands.

Notes

Out of the complete set of meta commands, several are defined with the 'Frame End' attribute set to TRUE by default. Defaulting several key meta commands to be frame end commands allows the system designer to quickly build meta scripts without having to worry about the framing mechanics used by the script engine. However, if they choose to have more direct control over command framing, they can easily change the *dwFlags* field of any command and enable or disable its XMC_MF_FRAMEEND attribute.

Motion Frames

A motion frame is a set of meta commands where only one, the end meta command, has its 'Frame End' attribute set to TRUE – the 'Frame End' attribute for all other commands is set to FALSE. The main purpose of the frame is to provide to the script engine on the target device a sequence of motion operations that can run in a reliable manner even if the data link between the host machine and device becomes intermittent or is clipped. The target device's script engine will only run a frame once it is received in its entirety.

Definition

Each frame is actually a contiguous set of elements within an array of XMC Meta Commands where the last element in the set has the 'Frame End' attribute set to TRUE. The next frame starts immediately after the last element in the previous frame.

Motion Scripts

A motion script is a set of meta commands that defines a sequence of basic motion operations to be run by the target device. Each script is made up of one or more frames of motion. When sending a script to the target device the data is sent a frame at a time. Upon receiving the script data, the target device will only run each frame of motion only after each complete frame has been received.

3 • General Use

In the meta scripting system, there are two steps that take place: a.) Building the scripts, and b.) Running the scripts. The following sections describe each of these in more detail.

Building Scripts

Building of scripts involves selecting from the set of meta commands available to build scripts of motion sequences that define the motion operations. The following diagram shows the details of this process.

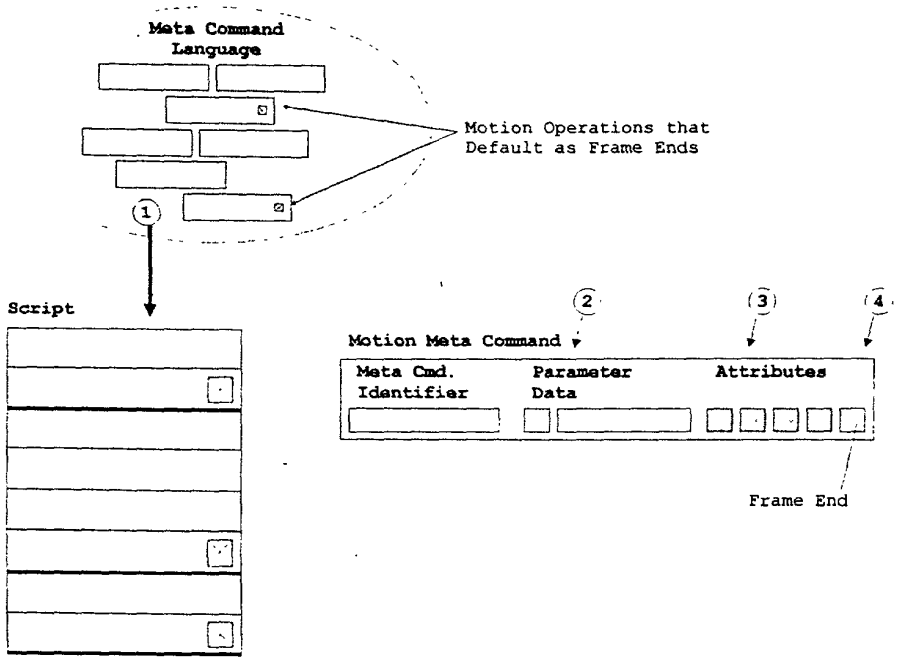
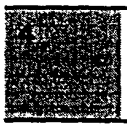


Figure 2 Building Script Data

The steps making up this process are as follows.

Step	Description
1	Select meta commands from the supported set of meta commands making up the scripting language to build each script in a manner that defines the sequence of motions to be run on the target device.
2	Set the parameter data for the meta command.
3	Set the general attributes (if any) supported by the meta command.



Set (or unset) the 'Frame End' attribute for the meta command. This step is optional for each meta command is defined with a default for this attribute which value depends on the command.

Once built, the script data can either be stored to a persistent medium and/or run on the target device.

Running Frames

To run the script data on the target device, the data must first be transferred to the device, which in turn then runs the data as each frame is received.

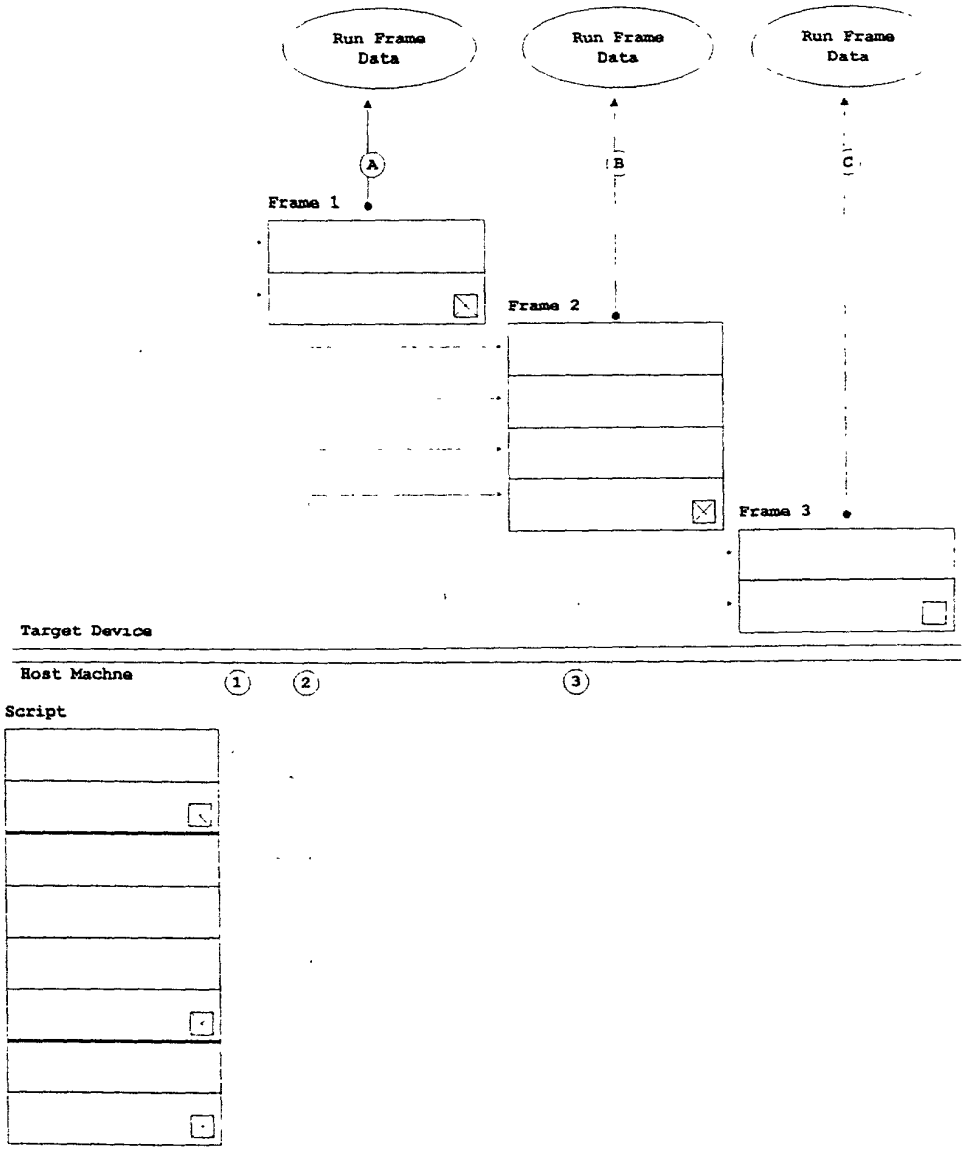
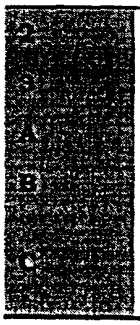


Figure 3 Running Script Data

The steps involved when running the script data are as follows.

Step	Description
1	Each frame is transferred to the device, in this case, the first frame in the script is transferred.



The second frame in the script is transferred.

The third frame in the script is transferred.

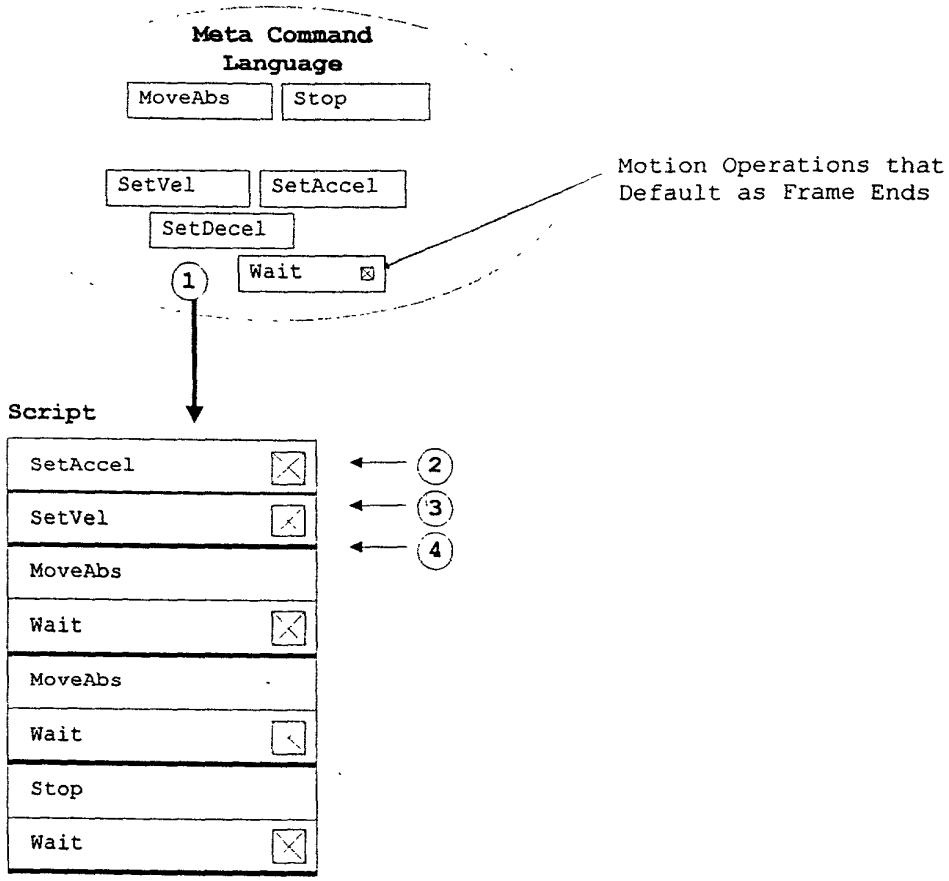
Once received, the first frame in the script is run.

Once received and the first frame is done running, the second frame is run.

Once received and the second frame is done running, the third frame is run.

4 • Example

In the following example, a script containing a sequence of motions is built. The motion sequence below is used to set the acceleration and velocity, make two absolute moves, and then stop all motion (if any).



The following steps were used to build this script.

Step	Description
	Meta commands are selected from the meta language and organized in the desired execution order in the script.
	All parameter data is entered for each meta command.
	All general attributes (if any) are set for each meta command.
	The 'Frame End' attribute is set on several commands to make the download/run process more efficient. In this example this attribute was set on the 'SetVel' and 'SetAccel' meta commands for their default

setting for the 'Frame End' attribute is FALSE. The 'Wait' command, on the other hand has a default setting of 'TRUE' for the device must wait for a motion to complete before completing most operations.

Running the Data

When running the data, the host sends each motion frame to the device starting with the first one. The device, in turn, runs each frame as it is received. The following download/run process takes pace with the motion data.

a.) Download Frame 1 - { SetAccel }

b.) Download Frame 2 - { SetVel }

c.) Download Frame 3 - { MoveAbs, Wait }

d.) Download Frame 4 - { MoveAbs, Wait }

e.) Download Frame 5 - { Stop, Wait }

During the download process, the device may start running each frame as it is received. The following table shows an example of how the download and run sequences may actually overlap.

Frame	Meta Commands	Host Machine	Target Device
1	{ SetAccel }	Download Frm 1	Waiting for input
2	{ SetVel }	Download Frm 2	Run Frm 1
3	{ MoveAbs, Wait }	Download Frm 3	Run Frm 2
4	{ MoveAbs, Wait }	Download Frm 4	Run Frm 3
5	{ Stop, Wait }	Download Frm 5	Pending on Frm 3

EXHIBIT 3

Table of Contents

1 • Overview

This document describes the interleaving general model and algorithms used to mix motion data with other types of media such as audio or video. The intent of mixing such data is to allow motion instructions to play in sync with other media. For example, motion instructions may direct an object to move in sync with a certain musical song giving the appearance that the object is dancing to the music played.

The following chapters make up this document:

Chapter 1 – Overview; this chapter.

Chapter 2 – Interleaving Model; describes how interleaving works and the packing methods used to build the target data stream.

Chapter 3 – General Algorithms; describes different packing and unpacking algorithms.

2 • Interleaving Model

Interleaving is the process of merging two data sources, from two different data types into a single data stream that is then transferred to the target player. The target player, in turn plays each data set concurrently as the data is received.

Interleaving is a technology designed to synchronize two data types so that when they are played the end results (ie musical sounds and motion driven movements) are synchronized with one another. For example, interleaving allows music data and motion instructions to be mixed in a manner that when played on a robotic device, the device dances in sync with the music.

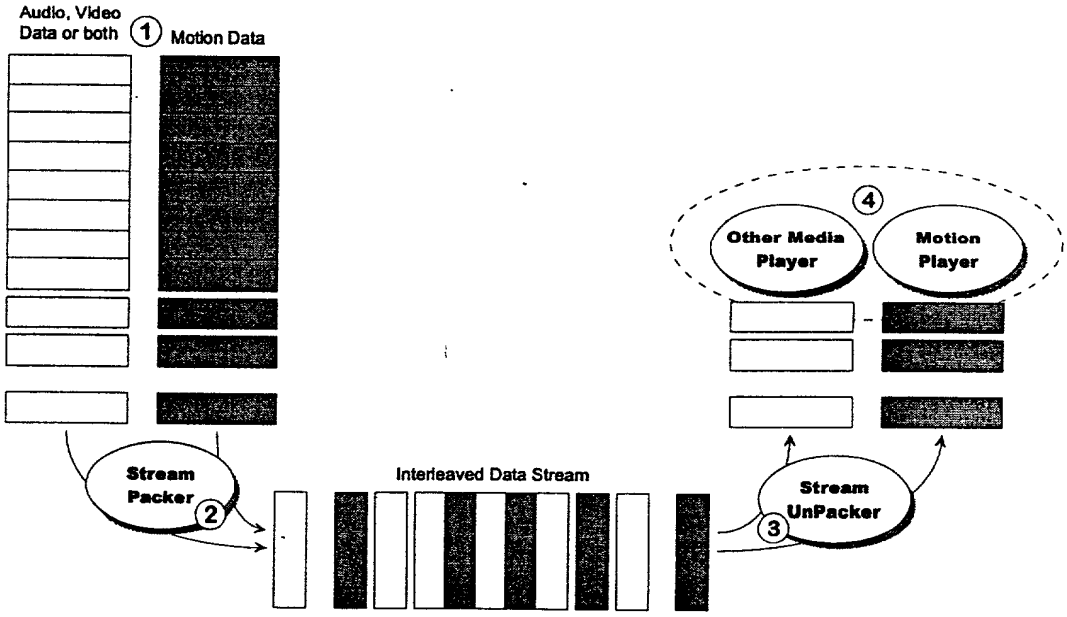


Figure 1 General Interleaving Model.

The following steps occur during the interleaving process.

Step	Description
1	The process starts with two data sources that are to be merged together.
2	Next, the Stream Packer takes each data source and packs each into the interleaved data stream using a packing algorithm to alternate the selection of data from each stream.
3	When used, the interleaved data stream is unpacked using the Stream Unpacker which is used to extract each data packet and pass each to the appropriate data player (or appropriate data player module in a single player unit). The data is passed to each player based on the data packet type. For example, the motion data is passed to the motion player, whereas the audio and/or video data is passed to the

	audio and video players respectively.
	The data is played concurrently and in sequence synchronizing the end results. For example, playing motion data at the same time as playing audio data causes the motions to be in sync with the audio sounds.

Packing Methods

The packing method, used by the Stream Packer, determines how the motion media is mixed with other media types such as audio or video data. There are four main methods used to mix the data during the packing process.

Time Based Packing

Time based packing is the process of selecting data from each data stream based on a pre-specified time quantum. A different time quantum may be used with each data stream.

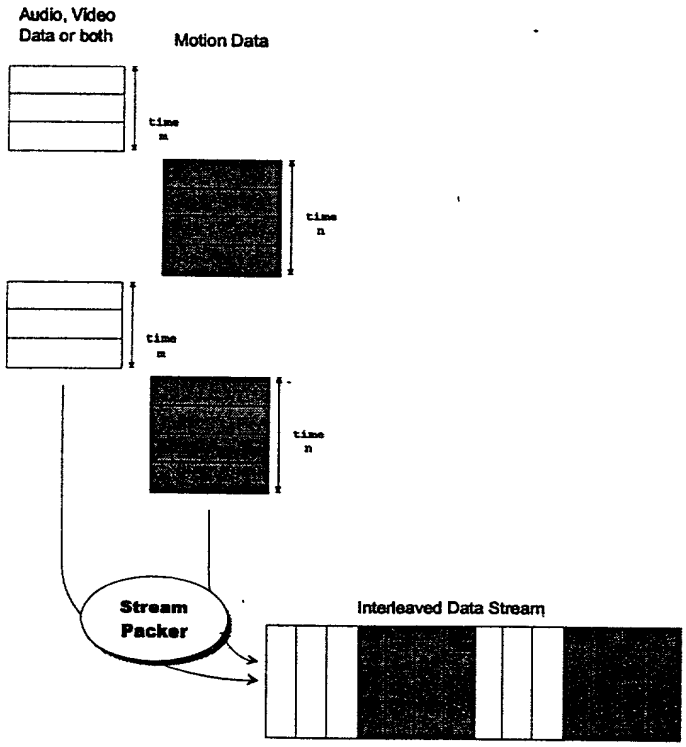


Figure 2 Time Based Packing

Upon selecting the data source to select packets from, the time quantum is reset to its pre-specified value associated with the data source. Packets are then pulled from the data source until the time quantum expires.

Packet Size Based Packing

Size-based packing is a packet selection method where packets are selected from the data source until a specified number of bytes is packed from the data source into the target interleaved data stream. Upon reaching or exceeding the specified size, packets are then selected from another data source.

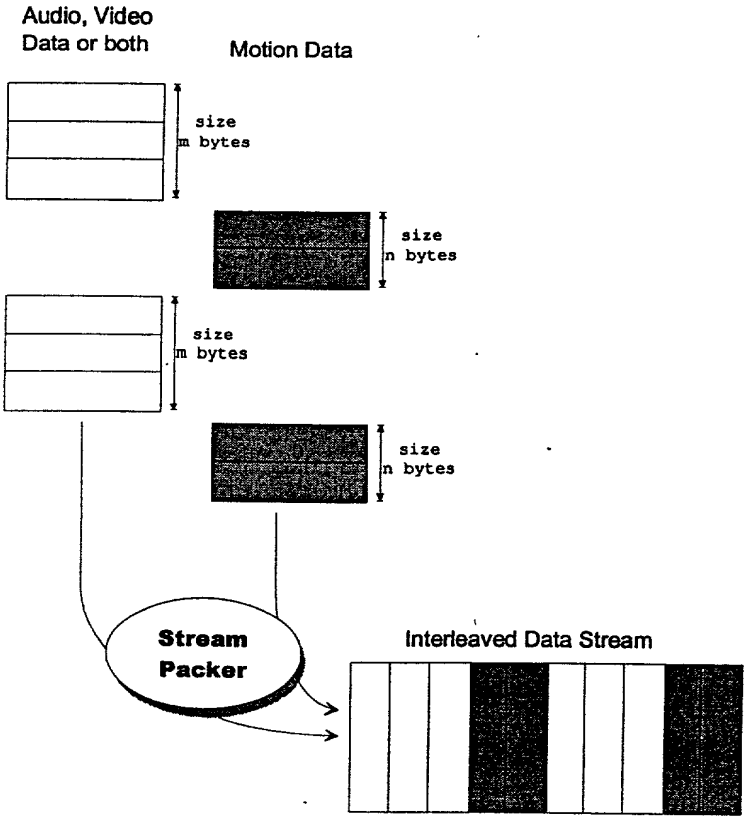


Figure 3 Size Based Packing.

Size based packing is used to ensure that each set of packets from each data source is packed in the target stream as a group of packets whose data size is at least a pre-specified size.

Packet Count Based Packing

When building the interleaved data stream with count based packing, a specified count of packets from the current data type (ie motion, audio, video, etc) are placed in the target interleaved data stream before adding packets from other data types. After switching to the new data type a specified count of packets from the new data type are placed in the data stream. A specified count of packets of each data type are placed into the target data stream until no data remains in any of the original data specific data streams.

Count-based packing is a packet selection method where packets are selected from the data source until a specified number of packets are packed from the data source into the target interleaved data stream. Upon reaching or exceeding the specified packet count, packets are then selected from another data source.

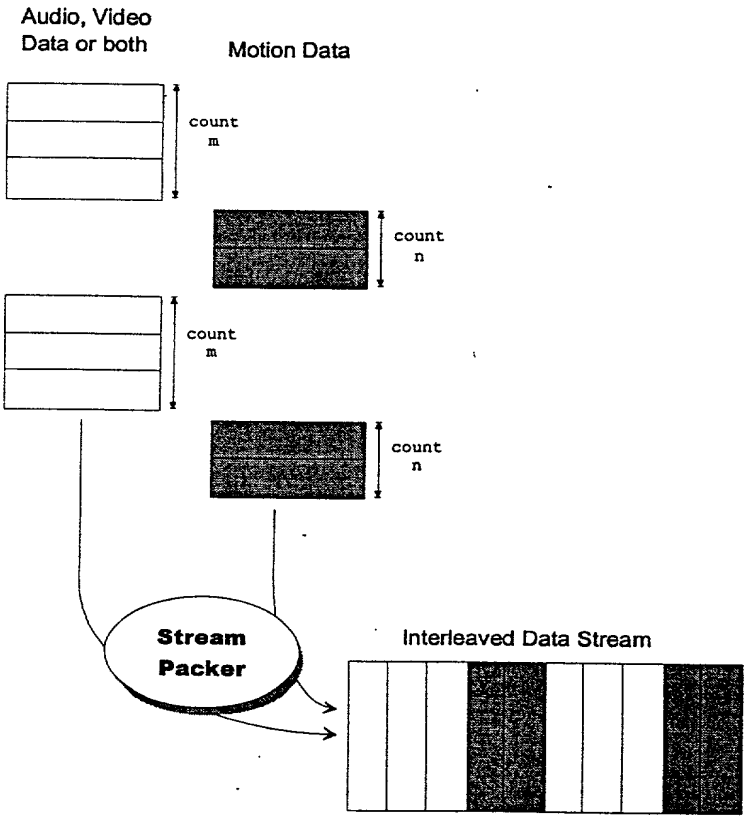


Figure 4 Count Based Packing.

Count based packing is used to ensure that a specific count of packets from each data source are grouped together in the target interleaved data stream.

Combination Packing

Combination packing is the use of a different packing algorithm for each data source. For example, when loading data from the motion data source size-based packing may be used, yet when loading data from the audio or video data source time-based packing may be used. The mix of different methods is optional and provided to help better synchronize the data.

EXHIBIT 4

Motion Network Models

XMC Motion Network Models

Author: Dave Brown
Create Date: October 31, 1999 Save Date: XXX 0, 0000 Print Date: November 1, 1999
Project: $/prjcmpnt/xmc/v.100/int/persdev/doc/_devpers/xmcmeta/des/network_models
Document: Document2
Description:
Revisions:

ROY-G-BIV®
Software for a spectrum of ideas™

Table of Contents

1 • Overview

When using the scripting & framing and live update and streaming technologies the host machine – target device relationship may take many forms. This document details out several of these relationships and how they may be used by the end user.

The general relationship is that of a host machine connected to a target device by a data link of some sort. Where the host is responsible for creating and storing data, the target device is responsible for consuming the data sent to it by the host across the data link.

With this organization, there are three main models:

- **Broadcasting** – this model is defined as a host machine sending data out enabling several devices to pick up the data and execute it. The data is sent out much in the same way that a radio station broadcasts a radio signal to many radio devices.
- **Request Brokering** – this model inverts the broadcast model in that data is only sent to each device after the device makes a data request. The host machine acts as a data broker in that it only sends data to a device when requested.
- **Autonomous Distribution** – this model is a mix of both the broadcast and request-brokering model in that each device plays the role of both the host machine and the target device. In this model each device is capable of broadcasting data to all other devices as well as broker data requests from each. Each device also plays the role of the data consumer in that each device executes the data sets that it either requests or broadcasts that it is *tuned-in* to execute.

This document describes each of these models along with the basic model. The following chapters make up the document.

- **Chapter 1 – Overview**; this chapter.
- **Chapter 2 – Basic Model**; describes the general relationship between the host machine, data link and target device.
- **Chapter 3 – Network Models**; describes several different uses of the basic model.

2 • Basic Model

The basic model involves a host machine connected to a target device where the host is responsible for creating and storing the data sets, the data link is responsible for transferring the data to the target device, and the target device is responsible for consuming the data by executing it. The basic model may optionally have a user interface on the host and/or target side to enable the end user to configure each.

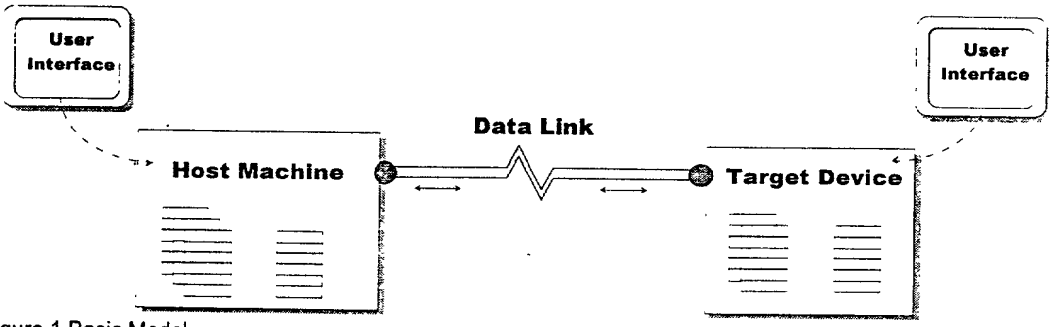


Figure 1 Basic Model.

It should be noted that even though the picture above shows both the host and target as separate physical machines, it is entirely possible that a single device provide both sets of functionality. This arrangement is used in one of the network models described later in this document.

Responsibility of each Item

Each of the items making up the basic model are described as follows:

- **Host Machine** – The host machine creates and stores each data set in either a live update or stream form. Data sets can be organized in script and frame format or just as a sequence of meta commands that describe intended operations for the target device to carry out. Even though the focus of this technology is on that of a sequence of *motion control* operations, the operations may also describe any operation used to control the target device such as the following:
    - Controlling and/or querying digital or analog IO lines on the device.
    - Controlling and/or querying a video camera, digital still camera or other vision-based sensor.
    - Controlling and/or querying any of a large array of sensors such as accelerometers, thermocouples, pressure transducers, etc.
    - Controlling and/or querying internal logic sequences (such as soft logic) or other algorithm running on the device.

- Controlling and/or querying motion control operations.

- **Data Link** – The data link is the medium across which the data is sent from the host machine to the target device. The data link may be in many different forms such as the following:
  - Ethernet based network link (or other physical wire based network such as TokenRing).
  - Wireless link (infrared, high bandwidth wireless or satellite).
  - Physical backplane (ISA or PCI bus). When host and target device are implemented as a single machine the back-plane acts as the data link between the logic implemented by each.

- **Target Device** – The target device is the consumer of the data. To consume the data the target device executes the set of logical machine instructions associated with each meta command making up the data set.

Data Channels

Each data link can be designated as a channel in its entirety, or be segmented into several channels. To segment a data link into a channel, each *packet* of data is marked with its channel number before sent across the data link. When received the target device can then discern whether or not the data item is actually on the data channel that it is currently listening to. If the entire data link is designated as a single channel, the marking of each data packet is not required.

Each packet of data is a set of one or more frames of meta commands along with additional packet attributes such as the data channel number or sequencing number for the packet.

3 • Network Models

There are several important uses of the basic model, namely: broadcasting, request brokering, and autonomous distribution. The following sections describe each of these in detail.

Broadcasting

Broadcasting is a variation of the basic model where the host machine sends data sets across one or many data links to one or many different devices simultaneously. The host machine may not even be aware that any devices are *listening* for data sets, but this is not required in the broadcast model. In this model, the host machine is mainly concerned with sending data sets out across the data link(s) without caring who is on the other end of those links.

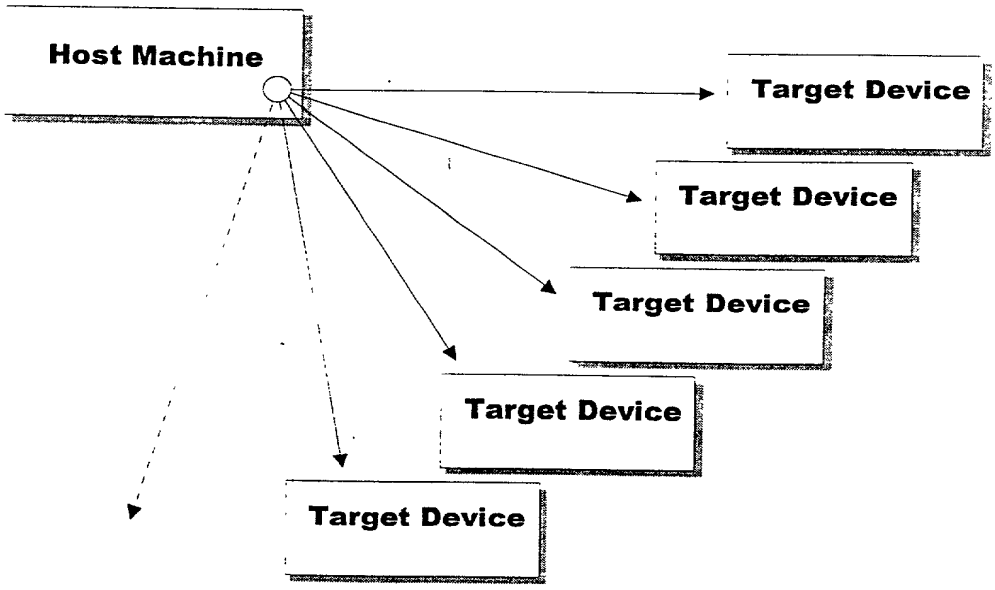


Figure 2 Broadcast Network Model

On the other side of the data link the target devices receive data that they are *tuned* to listen to. For example, each device may be tuned to listen to a single *channel* of data (i.e. a certain portion of a data link, or even just a specific data link chosen out of a set of data links). When listening to a data channel, the device then executes all data received by running all machine logic that it has associated with each meta command in the data set.

Request Brokering

In the request broker model, one or more target device request data updates from the host machine. Each request may occur simultaneously or at different times. The host machine waits idle until it receives a request for a data set or connection to a data channel. Once requested, the host machine associates the data set and/or data channel with the requesting device and begins transmitting data across the data link.

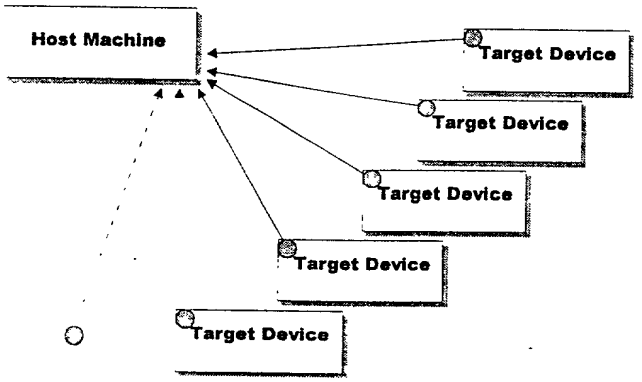


Figure 3 Request Brokering Network Model

Upon receiving the data requested, the target device runs each meta command within the data by executing the logical machine instructions that it has associated to each.

Autonomous Distribution

The autonomous distribution is a mix of the broadcast and request broker models in that each target device internally plays the role of both the host machine and target device. As host machine each device creates, stores and transmits data sets to other devices, and as a target device it either consumes its own data sets or other data sets received from other devices.

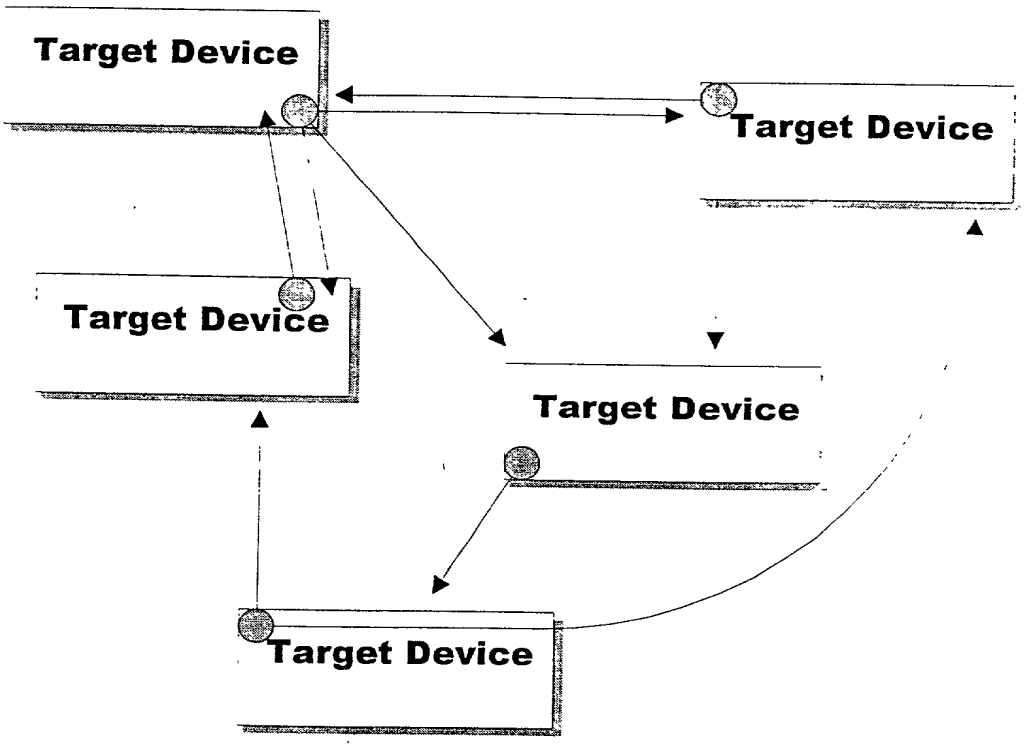


Figure 4 Autonomous Distribution

With this model target devices can work together as a community and be used to divide and conquer to break down a difficult task.

EXHIBIT 5

Table of Contents

1 • Overview

Data updating is the process of transferring data sets from the machine used to create and store the data set to a target device used to consume the data. This document details out the process that takes place when making these data transfers, or otherwise know as *live updates*.

Both the host machine and target device play integral roles in any interactive system. In an interactive system, the host is defined as the machine (or device) that creates and stores data sets that are to be later run on the target device. The target device (which may actually be he same host machine or a separate independent machine or device) is the consumer of the data sets. Consuming, or playing, the data is the process of executing the instructions or meta codes making up the data set to cause an event (either external or internal).

For example, the host machine may be used to generate frames of motion meta commands that make up the scripts defining desired behaviors for a particular target device. When received, the target device plays each frame of motion meta commands by executing (or running) the action associated with each thus causing motion actions to occur. In this example, one machine could perform both host and device activities or they could also be on separate machines, connected by a data link (i.e. tethered line, network connection, wireless connection, the Internet, etc).

The following chapters make up this document:

- **Chapter 1 – Overview**; this chapter.
- **Chapter 2 – Data Update Models**; describes the different data update models used to perform data updates.
- **Chapter 3 – Initiating Data Updates**; describes the different methods of initiating a data update session.

2 • Data Update Models

There are three variations of the general *live update* model used to transfer data from the host to the target device and involve the following: scheduling, requesting and streaming 'live update' data. The following sections detail out the general live update model and each of these three variations of it. The data transferred can take many forms including: only motion data (where the data is a sequence of basic motion control operations), a mix of motion and music data (where the motion data may choreographed to the music data), or motion, music and video data combined together.

Live Update

Live updating is the process of transferring data from a host (a location used to create and store data) to a target (a machine that is used to execute the data). The location for both host and target operations may be on the same machine or on separate machines as described below.

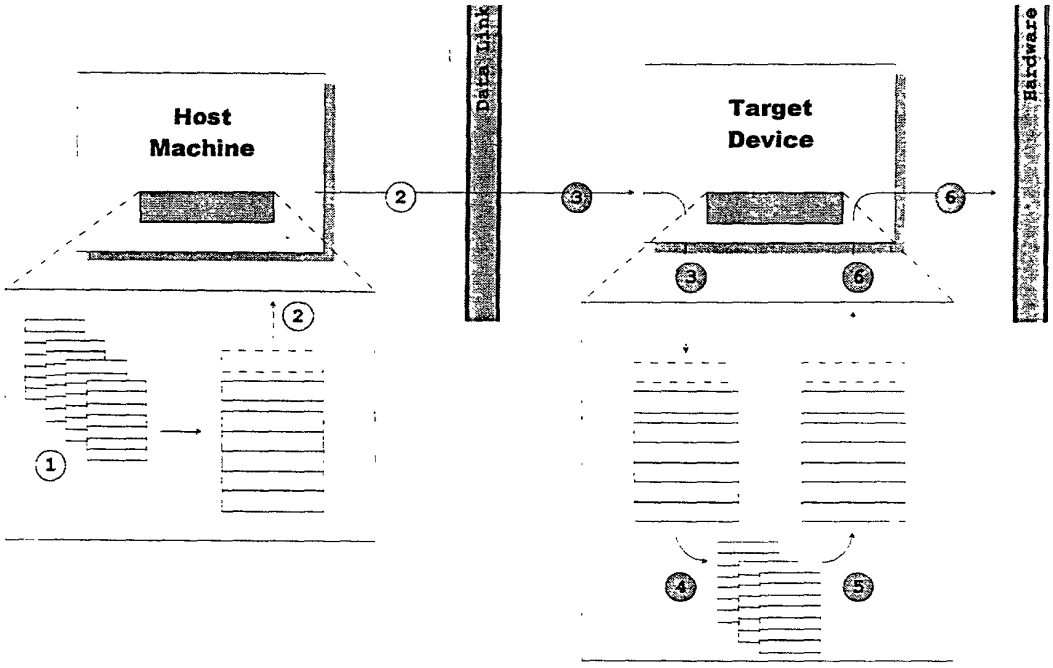


Figure 1 Live Update Process

The following steps take place during the live update process.

Step	Description
1	The first step in the live update process is that of the host machine

selecting from a library of scripts that are to be transferred to the target device.

2. Once selected, each frame of motion sequences (where a frame is defined as a set of motion meta commands, that describe basic motion operations, with the last motion meta command marked as the 'frame end' attribute). See 'Scripting and Framing' design document for details. It is possible, however not recommended to also send each script of motion data to the target device one meta command at a time. This is not recommended for doing so could place the target device in an unpredictable state if the data link connecting the host machine and target device is clipped.

3. The target device stores each frame received in a temporary location until all frames making up the script are received.

4. Once the script is received in its entirety, it is stored in a *ready* queue signifying that it is ready for execution.

5. When the script becomes the target script (i.e. the script selection logic of the device selects the script via a time quantum, programmed logic, or basic sequential order of each script) the device initiates executing the script.

6. To execute the script, the target device executes each **meta command** in the sequential order defined by the script. At this point the frames are not used. Execution of a meta command is the process of running the motion logic associated with each meta command.

Streaming

Streaming is the process of continually running through the live update process described in the previous section. Instead of sending scripts of data, the data set plays much like a continuous musical song, where frames of motion are continually sent to the target device, which in turn stores each frame received and runs each in the sequence received.

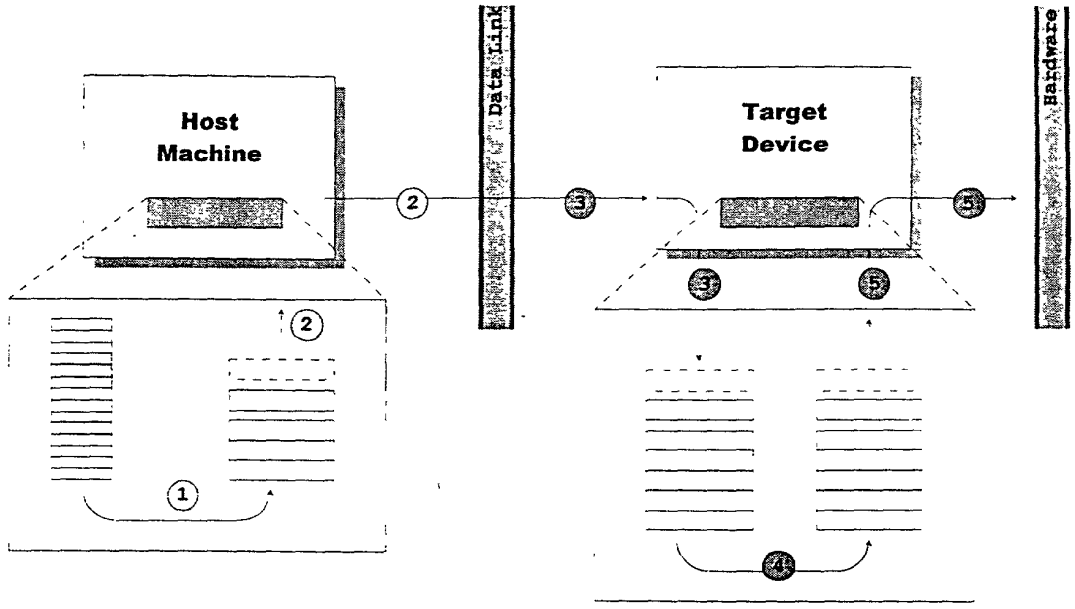


Figure 2 Streaming.

The following steps take place during the live update process.

Step	Description
1	When streaming, the host machine sends each frame of a large set of frames (or a set of frames that is continually being created or generated by the host machine) to the send queue.
2	Next, each frame (made up of one or more meta commands that can only be run by the target device as a set) is sent to the target device.
3	Upon receiving each frame, the target device adds them to a receive queue. Note, it is possible that the frames received may be received out of sequence. In such a case, the frames are inserted into the receive queue in a manner that maintains their original sequence sent by the host machine. More than often the sequencing is performed by the underlying network protocol, but this is not required for the host and target device may implement their own sequencing mechanism.
4	Unlike the like update method that waits until a full script of data is received before continuing, each frame is immediately passed to the target device output queue in preparation for execution.

5. Each **meta command** within the output queue is executed by running the motion logic associated with each command.

3 • Initiating Data Updates

Data updates can be initiated either by the host machine, called *Push Updates*, or by the target device, called *Pull Updates*. The following sections describe each of these methods of initiating either a live or streaming data update session.

Push Update

Push updates are a variation of the *data update* models where the host only initiates updates after pre-determined events occur (external or internal). Certain time intervals, an internal logical sequence or an external input are all example events that could be associated with live updates that are fired to the target device when encountered.

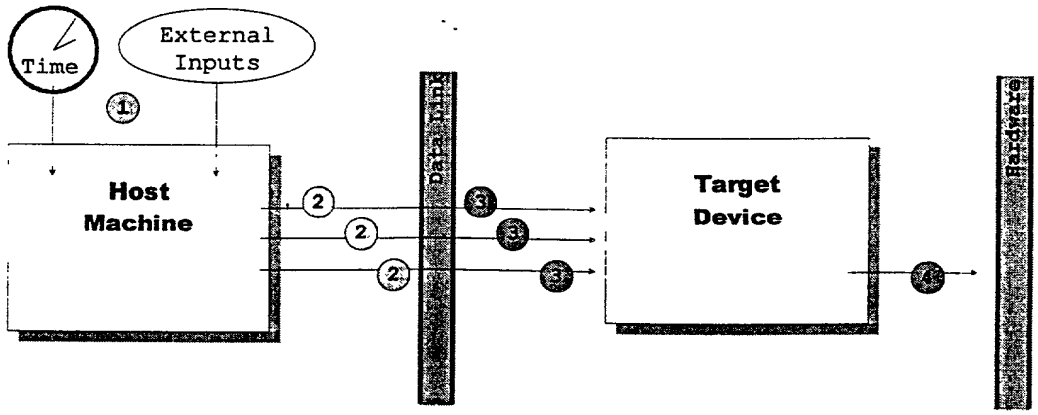
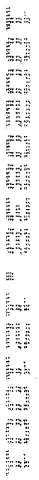


Figure 3 Push Updating.

The following steps take place during the live update process.

Step	Description
1	Either an external event, a specified time interval or some other internal logical algorithm running on the host machine initiates the Push update.
2	Once the update data event occurs, the host machine starts sending frames of data (either in a live update or streaming form) to the target device connected via the data link.
3	Upon receiving the update data, the target device buffers the data treating it as either a live update or as a stream of data.
4	As data becomes ready internally, the target device begins executing each meta command by running the motion logic associated with each, which eventually manipulates, monitors and/or controls the underlying hardware.



Pull Updates

Pull updates are a version of the *live update* model where the target, instead of the host, initiates the live update process. After encountering a pre-determined event (or set of events) the device requests for a live update from the host machine. Upon receiving such notification, the host machine then runs through the update process to transfer the requested data back to the target device.

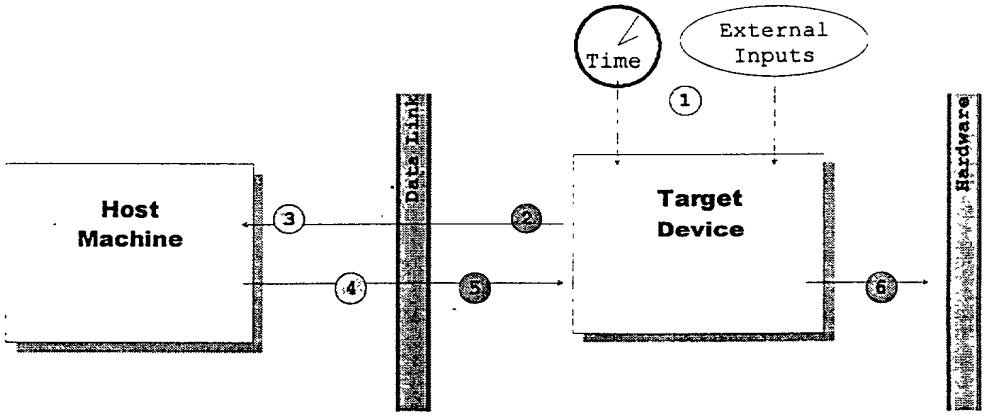


Figure 4 Pull Updating.

The following steps take place during the live update process.

Step	Description
1	Unlike Push updates, the target device initiates a Pull data update after either an external event occurs, a specified time increment passes or an internal logical algorithm dictates.
2	When the data update trigger event occurs, the target device fires a *data update request* to the host machine to initiate the update process.
3	Upon receiving the data update request the host machine begins preparing the data for either a live update or data streaming data transfer.
4	When ready the frames of motion meta commands are transferred to the target device across the data link connection.
5	When received, the device process the data much in the same way that it does in the Push data update.
6	The data received is run either as a live update or a stream by executing each meta command making up the frames of data. Executing the meta command consists of running the motion logic associated with each meta command, which manipulates, monitors and/or controls the target hardware.

United States Patent & Trademark Office

Office of Initial Patent Examination -- Scanning Division

Application deficiencies found during scanning:

☐ Page(s)_____of_____ were not present for scanning.
(Document title)

☐ Page(s)_____of_____were not present for scanning.
(Document title)

☑ ***Scanned copy is best available.*** Some exibit pages are dark.